Pattern Matching part one

Suffix Trees

Benjamin Sach
**Exact pattern matching**

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[ T = \text{abcbaabcaab} \]
\[ P = \text{aba} \]

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

$T$:

```
  a b c b a b a b a c a b a
```

$P$:

```
  4 a b a
```

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
**Exact pattern matching**

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[ T \]

\[ P \]

\[ n \]

\[ m \]

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th>$T$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$b$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
<th>$c$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
</tbody>
</table>

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

Goal: Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

![Diagram showing text string $T$ and pattern string $P$]

**Goal**: Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i+j]$

*(our strings are zero-indexed)*

- A naive algorithm takes $O(nm)$ time
**Exact pattern matching**

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th></th>
<th>$b$</th>
<th>$c$</th>
<th>$b$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
<th>$c$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
</tr>
<tr>
<td>$P$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*our strings are zero-indexed*

- A naive algorithm takes $O(nm)$ time
- Many $O(n)$ time algorithms are known (for example KMP)
Preprocess a text string $T$ (length $n$) to answer pattern matching queries...
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

After preprocessing, a query is a pattern $P$ (length $m$),

$$T = a \ b \ c \ b \ a \ b \ a \ b \ a$$

$$P = a \ b \ a$$
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

After preprocessing, a query is a pattern $P$ (length $m$), the output is a list of all matches in $T$. 
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries... 

After preprocessing, a query is a pattern $P$ (length $m$),

the output is a list of all matches in $T$.

e.g. 4, 6, 10
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

After preprocessing, a query is a pattern $P$ (length $m$), 
the output is a list of all matches in $T$.

- A naive algorithm takes $O(n)$ query time (using KMP)
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

After preprocessing, a query is a pattern $P$ (length $m$),

the output is a list of all matches in $T$.

e.g. 4, 6, 10

- A naive algorithm takes $O(n)$ query time (using KMP)

- We want a query time which depends only on $m$ and $\text{occ}$
  - $\text{occ}$ is the number of occurrences (matches)
Text indexing

Preprocess a text string \( T \) (length \( n \)) to answer pattern matching queries...

\[
\begin{array}{cccccccc}
T & a & b & c & b & a & b & a & c & a & b & a \\
\end{array}
\]

After preprocessing, a **query** is a pattern \( P \) (length \( m \)),

\[
\begin{array}{ccc}
P & a & b & a \\
\end{array}
\]

the output is a list of all matches in \( T \).

- A naive algorithm takes \( O(n) \) query time (using KMP)
- We want a query time which depends only on \( m \) and \( \text{occ} \)
  - \( \text{occ} \) is the number of occurrences (matches)
- We also want \( O(n) \) space and fast preprocessing (prep.) time
The atomic suffix tree

\[ T \]

\[
\begin{array}{cccccc}
  b & a & n & a & n & a & s \\
\end{array}
\]

\[
 n 
\]
The atomic suffix tree

\[ T \]

```
  b a n a n a s  
    \  n \  
  b a n a n a s  
    a n a n a s  
    n a n a s  
    a n a s  
    n a s  
    a s  
    s  
```
The atomic suffix tree

<table>
<thead>
<tr>
<th>T</th>
<th>banananas</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
</tr>
<tr>
<td>banananas</td>
<td></td>
</tr>
<tr>
<td>ananas</td>
<td></td>
</tr>
<tr>
<td>nanas</td>
<td></td>
</tr>
<tr>
<td>nas</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td></td>
</tr>
</tbody>
</table>

suffixes: s, as, nas, anas, banananas, banana

suffix tree:

- Root node labeled 's'
- Nodes labeled with substrings of 'banananas'
- Edges labeled with characters of 'banananas'
- Direct edges to child nodes
The atomic suffix tree

```
T  bananas
    bana
        ana
            na
                    a
                                      s
```

suffixes
The atomic suffix tree

$T$  

- $b a n a n a s$
- $b a n a n a s$
- $a n a n a s$
- $n a n a s$
- $a n a s$
- $a s$
- $s$

SUFFIXES

SUFFIX TREE
The atomic suffix tree

T

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>n</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>n</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

suffix tree
The atomic suffix tree
The atomic suffix tree

T

bananas

suffixes

bananas

ananas

anas

nas

as

s

suffix tree

n

a

s

b

n

a

s

a

n

a

s

a

s

a

s

a

s

a

s

a

s

a

s

a

s

a

s

a

s
The atomic suffix tree

$T$

$banana$ $s$

suffix tree

$\textit{suffixes}$

$banana$

$anana$

$anana$ $n$

$s$
The atomic suffix tree

T

The suffix tree

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>a</td>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

suffixes
The atomic suffix tree

T

banana

banana

anana

nana

anap

as

s

Suffixes

Suffix tree
The atomic suffix tree

The atomic suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree

suffix tree
The suffix tree contains every suffix of $T$ as a root to leaf path.
• The suffix tree contains every suffix of $T$ as a root to leaf path
• Every edge is labelled with a character from $T$
The suffix tree contains every suffix of $T$ as a root to leaf path.

Every edge is labelled with a character from $T$.

No two edges leaving the same node have the same label.
• The suffix tree contains every suffix of $T$ as a root to leaf path
• Every edge is labelled with a character from $T$
• No two edges leaving the same node have the same label
• Each leaf corresponds to a suffix (so there are $n$ leaves)
Searching in an atomic suffix tree

$T \quad b\,a\,n\,a\,n\,a\,s$

---

$n$

---

Diagram of an atomic suffix tree with nodes labeled with letters and numbers indicating positions in the tree. The text is part of a larger discussion on searching techniques in suffix trees.
Searching in an atomic suffix tree

How do you find a pattern?
Searching in an atomic suffix tree

How do you find a pattern?
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
How do you find a pattern?

start at the root and walk down the tree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in an atomic suffix tree

**T**

\[
\begin{array}{c}
 b \\
\hline
 a \\
\hline
 n \\
\hline
 a \\
\hline
 n \\
\hline
 s
\end{array}
\]

**P**

\[
\begin{array}{c}
 a \\
\hline
 n \\
\hline
 a \\
\hline
 m
\end{array}
\]

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

$T \quad b\ a\ n\ a\ n\ a\ s$

$P \quad a\ n\ a$

$P' \quad n\ a\ b$

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

\[ T = \textbf{b a n a n a s} \]

\[ P = \textbf{a n a} \]

\[ P' = \textbf{n a b} \]

How do you find a pattern?

- start at the root and walk down the tree
- \ldots matches occur at the leaves of the subtree
Searching in an atomic suffix tree

\[ T \begin{array}{cccccc}
  b & a & n & a & n & a \\
  n & & & & & \\
\end{array} \]

How do you find a pattern?

- start at the root and walk down the tree
- …matches occur at the leaves of the subtree

\[ P \begin{array}{ccc}
  a & n & a \\
  m & & \\
\end{array} \checkmark \]

\[ P' \begin{array}{ccc}
  n & a & b \\
\end{array} \times \]
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time

(we’ll worry about outputting the matches later)
Searching in an atomic suffix tree

How do you find a pattern?

- start at the root and walk down the tree
- matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time

(we’ll worry about outputting the matches later)
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time

(we’ll worry about outputting the matches later)

---

**WARNING!** How long does it take to find the correct child?

There could be $n$ edges here!

In this lecture we assume the alphabet size is a constant

This may be fine in some applications

(English text or DNA for example)

We can remove the assumption via the magic of hashing
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
   ... matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time

(we’ll worry about outputting the matches later)
how large is the atomic suffix tree?

There are at most $n$ leaves
how large is the atomic suffix tree?

There are at most $n$ leaves

that's good right?
how large is the atomic suffix tree?

There are at most $n$ leaves

that's good right?

Unfortunately there can be *lots* of internal nodes
how large is the atomic suffix tree?

There are at most $n$ leaves
that's good right?

Unfortunately there can be lots of internal nodes
how large is the atomic suffix tree?

There are at most $n$ leaves

that's good right?

Unfortunately there can be *lots* of internal nodes
how large is the atomic suffix tree?

There are at most $n$ leaves

That's good right?

Unfortunately there can be lots of internal nodes

7 characters
how large is the atomic suffix tree?

There are at most $n$ leaves

that's good right?

Unfortunately there can be lots of internal nodes

7 characters  23 nodes
how large is the atomic suffix tree?

There are at most $n$ leaves

that's good right?

Unfortunately there can be lots of internal nodes

7 characters   23 nodes   that’s not so bad, right?
how large is the atomic suffix tree?
how large is the atomic suffix tree?

$T \begin{array}{c|c}
2 & \\ \\
a & b \\
\end{array}$
how large is the atomic suffix tree?
how large is the atomic suffix tree?

$T \begin{array}{c|c} a & b \\ \hline \end{array}$

4 nodes
how large is the atomic suffix tree?

4 nodes

9 nodes
how large is the atomic suffix tree?

- For \( T = a b \), there are 4 nodes.

- For \( T = a a b b \), there are 9 nodes.

- For \( T = a a a b b b \), there are 16 nodes.
how large is the atomic suffix tree?

- $T \begin{array}{ll} a & b \end{array}$ with 4 nodes
- $T \begin{array}{lll} a & a & b \end{array}$ with 9 nodes
- $T \begin{array}{lll} a & a & a & b & b & b & b & b \end{array}$ with 25 nodes
how large is the atomic suffix tree?

T

<table>
<thead>
<tr>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

4 nodes

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

9 nodes

<table>
<thead>
<tr>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

16 nodes

<table>
<thead>
<tr>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

25 nodes

<table>
<thead>
<tr>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

36 nodes
An atomic suffix tree can have 
\[ ((n/2) + 1)^2 \] nodes
how large is the atomic suffix tree?

An atomic suffix tree can have \(((n/2) + 1)^2\) nodes

this is far too big!
Why is the atomic suffix tree so big?
Why is the atomic suffix tree so big?

because it has long paths like this one

$T = \text{banana}$

Compacted suffix trees
Why is the atomic suffix tree so big?

Main Idea replace each non-branching path with a single edge
Compacted suffix trees

Why is the atomic suffix tree so big?

Main Idea replace each non-branching path with a single edge

- edges are now labelled with substrings
Compacted suffix trees

Why is the atomic suffix tree so big?

Main Idea replace each non-branching path with a single edge
- edges are now labelled with substrings
  (instead of single characters)
Compacted suffix trees

Main Idea replace each non-branching path with a single edge
- edges are now labelled with substrings
  (instead of single characters)
Main Idea replace each non-branching path with a single edge
- edges are now labelled with substrings
  (instead of single characters)

There are at most $n$ leaves
Compacted suffix trees

Main Idea replace each non-branching path with a single edge
- edges are now labelled with substrings
  (instead of single characters)

- There are at most $n$ leaves
- Every internal node has two or more children
Main Idea replace each non-branching path with a single edge

- edges are now labelled with substrings

(instead of single characters)
Compacted suffix trees

Main Idea: replace each non-branching path with a single edge
- edges are now labelled with substrings
  (instead of single characters)

There are at most $n$ leaves
Every internal node has two or more children

so there are $O(n)$ edges
don’t the edges take up lots of space?
Compacted suffix trees

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
</table>

- There are at most $n$ leaves
- Every internal node has two or more children

so there are $O(n)$ edges

don’t the edges take up lots of space?

we only store the end points

**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

(instead of single characters)
Compacted suffix trees

Main Idea replace each non-branching path with a single edge

- edges are now labelled with substrings

(instead of single characters)

- There are at most $n$ leaves
- Every internal node has two or more children

so there are $O(n)$ edges

don’t the edges take up lots of space?

we only store the end points

we actually store $(4, 6)$
Compacted suffix trees

$T = \text{bananas}$

Diagram of compacted suffix tree with nodes labeled 0, 1, 2, 3, 4, 5, 6.
Compacted suffix trees

$T = \text{bananas}$

Compacted Suffix Tree of $T$

Diagram of a compacted suffix tree with nodes labeled from 1 to 6, showing the structure of the suffix tree for the string "bananas".
Compacted suffix trees

$T = \text{bananas}$

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
Compacted suffix trees

\[ T \quad \text{bananas} \]

Compacted Suffix Tree of \( T \)

- A rooted tree with \( n \) leaves
- Every internal node has two or more children
Compacted suffix trees

$T$: bananas

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
Compacted Suffix Trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character

$T$: bananas

Diagram of compacted suffix tree with leaves and internal nodes labelled with substrings.
Compacted suffix trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
Compacted Suffix Trees

**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

**Example:***

[T: b a n a n a s]
Compacted Suffix Trees

**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space
Compacted suffix trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Sanity Check

Does the compacted suffix tree always exist?

Uses $O(n)$ space
Compacted Suffix Trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Sanity Check

Does the compacted suffix tree always exist?

- $T = b\ b$

Uses $O(n)$ space

this doesn't have $n$ leaves
Compacted suffix trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Sanity Check

Does the compacted suffix tree always exist?

$T \quad b\ b$

- This doesn't have $n$ leaves

$T \quad b\ b\ $

- This has $n$ leaves

Uses $O(n)$ space
Compacted suffix trees

$T$: bananas

- **Compacted Suffix Tree** of $T$
  - A rooted tree with $n$ leaves
  - Every internal node has two or more children
  - Every edge is labelled with a substring
  - No two edges leaving the same node have the same first character
  - Each leaf is labelled with a location in $T$
  - Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space.
Compacted suffix trees

Step one: Add a $\$ $ (unique symbol) to $T$

$T \quad b \quad a \quad n \quad a \quad n \quad a \quad s$

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
</tr>
</tbody>
</table>

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space
Compacted suffix trees

**Step one:** Add a $\$$(unique symbol) to $T$

$T = \text{bananas} \$$(unique symbol)$$

**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space
Compacted suffix trees

Step one: Add a $ (unique symbol) to $T$

$T = \begin{array}{ccccccc} b & a & n & a & n & a & s \end{array}$

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space
Compacted suffix trees

**Step one:** Add a $\$$(unique symbol) to $T$

$T = b\,a\,n\,a\,n\,a\,s\,\$$

**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space

This is normally just called a suffix tree
Searching in a compacted suffix tree

\[ T \begin{array}{cccccc} b & a & n & a & n & a \end{array} \]
Searching in a compacted suffix tree

How do you find a pattern?
Searching in a compacted suffix tree

How do you find a pattern?

$T$: bananas$

$P$: anana

$\text{How do you find a pattern?}$
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

remember that an edge is actually stored as a pair we’re actually looking in \( T \)
Searching in a compacted suffix tree

**T**

```
bananas$
```

**P**

```
anna
```

---

**How do you find a pattern?**

start at the root and walk down the tree
Searching in a compacted suffix tree

**T**

```
| b | a | n | a | n | a | s | $ |
```

**P**

```
| a | n | a | n | a |
```

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

**How do you find a pattern?**

- start at the root and walk down the tree
- \ldots matches occur at the leaves of the subtree

**Diagram:**

- The text `T` is `banana$`
- The pattern `P` is `anna`
- The prefix `P'` is `na`

- The tree structure shows the compacted suffix tree with nodes labeled with substrings of `T` and `$` indicating the end of a substring.
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

$T$: 
\[ b\ a\ n\ a\ n\ a\ s\ ]$

$P$: 
\[ a\ n\ a\ n\ a\ ]$

$P'$: 
\[ n\ a\ ]$

How do you find a pattern?

start at the root and walk down the tree

…matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

…matches occur at the leaves of the subtree
Searching in a compacted suffix tree

**T**

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>n</th>
<th>a</th>
<th>s</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**P**

<table>
<thead>
<tr>
<th>a</th>
<th>n</th>
<th>a</th>
<th>n</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**P'**

<table>
<thead>
<tr>
<th>n</th>
<th>a</th>
</tr>
</thead>
</table>

---

**How do you find a pattern?**

- Start at the root and walk down the tree.
- Matches occur at the leaves of the subtree.

---

**O(occ)** because it has occ leaves (and each internal node has at least two children).
Searching in a compacted suffix tree

**T**

```
baananas$
```

**P**

```
ananana
```

**P'**

```
nan
```

---

*How do you find a pattern?*

1. Start at the root and walk down the tree.
2. Matches occur at the leaves of the subtree.

**O(occ)** because it has occ leaves (and each internal node has at least two children).

We can find all the matches in $O(m + occ)$ time (by looking at the whole subtree).
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree 
  \((\text{as if you were matching a pattern})\)
- Add a new edge and leaf for the new suffix 
  \((\text{this may require you to break an edge in two})\)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  \[(as \ if \ you \ were \ matching \ a \ pattern)\]

- Add a new edge and leaf for the new suffix
  \[(this \ may \ require \ you \ to \ break \ an \ edge \ in \ two)\]
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

you should never actually do it like this
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  \( (as \ if \ you \ were \ matching \ a \ pattern) \)

- Add a new edge and leaf for the new suffix
  
  \( (this \ may \ require \ you \ to \ break \ an \ edge \ in \ two) \)

\( T \)

\[
\begin{array}{c}
\text{b a n a n a s} \\
\hline
\text{n} \\
\text{b a n a n a s} \\
\text{a n a n a s} \\
\text{n a n a s} \\
\text{a n a s} \\
\text{n a s} \\
\text{a s} \\
\text{s} \\
\text{$} \\
\end{array}
\]
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  \((as \ if \ you \ were \ matching \ a \ pattern)\)

- Add a new edge and leaf for the new suffix
  
  \((this \ may \ require \ you \ to \ break \ an \ edge \ in \ two)\)

\textbf{you should} \textit{never actually do it like this}
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

you should *never* actually do it like this
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

you should never actually do it like this

we actually store this as (0, 7)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  \textit{(as if you were matching a pattern)}

- Add a new edge and leaf for the new suffix
  \textit{(this may require you to break an edge in two)}

\begin{itemize}
  \item you should \textit{never actually} do it like this
\end{itemize}
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  \[\text{(as if you were matching a pattern)}\]

- Add a new edge and leaf for the new suffix
  \[\text{(this may require you to break an edge in two)}\]
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

You should never actually do it like this.

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  \(\text{as if you were matching a pattern}\)

- Add a new edge and leaf for the new suffix
  
  \(\text{this may require you to break an edge in two}\)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

you should never actually do it like this
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

- Insert the suffixes one at a time (longest first)
  - Search for the new suffix in the partial suffix tree
    (as if you were matching a pattern)
  - Add a new edge and leaf for the new suffix
    (this may require you to break an edge in two)

You should never actually do it like this.
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

You should never actually do it like this.
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

you should *never actually do it like this*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix
  
  *(this may require you to break an edge in two)*

You should *never actually do it like this*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

you should never actually do it like this
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

• Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

• Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

you should never actually do it like this
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

```
T = bananas$

0  ✓
1  ✓
2  ✓
3  ✓
4  ✓
5  ✓
6  ✓
7  ✓

ananas$ was stored as (1, 7)
```

you should \textit{never} actually do it like this
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  \(\text{(as if you were matching a pattern)}\)

- Add a new edge and leaf for the new suffix
  \(\text{(this may require you to break an edge in two)}\)

- you should \textit{never} actually do it like this
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix (this may require you to break an edge in two)

You should never actually do it like this.
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  \[\text{(as if you were matching a pattern)}\]
- Add a new edge and leaf for the new suffix
  \[\text{(this may require you to break an edge in two)}\]
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

you should never actually do it like this
Naively constructing a compacted suffix tree

- Insert the suffixes one at a time (longest first)
  - Search for the new suffix in the partial suffix tree (as if you were matching a pattern)
  - Add a new edge and leaf for the new suffix (this may require you to break an edge in two)

Never actually do it like this.

You should never actually do it like this.
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  
  (this may require you to break an edge in two)

You should never actually do it like this.
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  \textit{(as if you were matching a pattern)}

- Add a new edge and leaf for the new suffix
  
  \textit{(this may require you to break an edge in two)}
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  \(\text{(as if you were matching a pattern)}\)

- Add a new edge and leaf for the new suffix
  \(\text{(this may require you to break an edge in two)}\)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

Never actually do it like this
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

You should never actually do it like this.
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)
  - Search for the new suffix in the partial suffix tree
    (as if you were matching a pattern)
  - Add a new edge and leaf for the new suffix
    (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  \[\text{(as if you were matching a pattern)}\]

- Add a new edge and leaf for the new suffix

\[\text{(this may require you to break an edge in two)}\]
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

This takes $O(n)$ time per suffix...
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  
  (this may require you to break an edge in two)

This takes $O(n)$ time per suffix... so $O(n^2)$ time in total

\[ T \]

```
  b a n a n a s $
  n

  b a n a n a s $
  a n a n a s $
  n a n a s $
  a n a s $
  n a s $
  a s $
  s $
  $ 
```

\[
\begin{array}{c}
\text{suffixes} \\
0 \checkmark \\
1 \checkmark \\
2 \checkmark \\
3 \checkmark \\
4 \checkmark \\
5 \checkmark \\
6 \checkmark \\
7 \checkmark \\
\end{array}
\]

- You should never actually do it like this
The (compacted) suffix tree of a (length $n$) text uses $O(n)$ space.

Finding all matches of a pattern $P$ of length $m$ takes $O(m + \text{occ})$ where $\text{occ}$ is the number of matches.

Suffix trees can be built in $O(n)$ time, but we have only seen the $O(n^2)$ time method. You should actually do it like this (or build a suffix array instead).

We assumed that the alphabet contained a constant number of symbols.
Multiple text indexing

How can we index multiple texts?
Multiple text indexing

How can we index multiple texts?

$T_1 \quad \text{banana\,n\,a\,s}\,\$  \quad n_1$

$T_2 \quad \text{apple\,s\,\&}  \quad n_2$

two distinct unique symbols
Multiple text indexing

How can we index multiple texts?
How can we index multiple texts?
How can we index multiple texts?
How can we index multiple texts?

- *Build a generalised suffix tree in* \(O(n_1 + n_2)\) *space*
How can we index multiple texts?

- Build a generalised suffix tree in $O(n_1 + n_2)$ space
- Using the linear time method (which we omitted), this takes $O(n_1 + n_2)$ time
How can we index multiple texts?

- **Build a generalised suffix tree in** \( O(n_1 + n_2) \) **space**
- **Using the linear time method (which we omitted), this takes** \( O(n_1 + n_2) \) **time**
- **Finding all matches of a pattern** \( P \) **of length** \( m \) **still takes** \( O(m + \text{occ}) \) **time**
  
  *where \( \text{occ} \) is the number of matches*
The suffix array - a sneak preview

$T \quad b \quad a \quad n \quad a \quad n \quad a \quad s

n
The suffix array - a sneak preview

<table>
<thead>
<tr>
<th>T</th>
<th>b a n a n a s</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>b a n a n a s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a n a n a s</td>
</tr>
<tr>
<td>2</td>
<td>n a n a s</td>
</tr>
<tr>
<td>3</td>
<td>a n a s</td>
</tr>
<tr>
<td>4</td>
<td>n a s</td>
</tr>
<tr>
<td>5</td>
<td>a s</td>
</tr>
<tr>
<td>6</td>
<td>s</td>
</tr>
</tbody>
</table>
The suffix array - a sneak preview

$T$:

\[ b\ a\ n\ a\ n\ a\ s \]

suffix:

0: \[ b\ a\ n\ a\ n\ a\ s \]
1: \[ a\ n\ a\ n\ a\ s \]
2: \[ n\ a\ n\ a\ s \]
3: \[ a\ n\ a\ s \]
4: \[ n\ a\ s \]
5: \[ a\ s \]
6: \[ s \]
The suffix array - a sneak preview

$T \quad b \, a \, n \, a \, n \, a \, s$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$b , a , n , a , n , a , s$</td>
<td>$a , n , a , n , a , s$</td>
<td>$n , a , n , a , s$</td>
<td>$a , n , a , s$</td>
<td>$n , a , s$</td>
<td>$a , s$</td>
<td>$s$</td>
</tr>
</tbody>
</table>
The suffix array - a sneak preview

Sort the suffixes lexicographically

T | n |
---|---|
 0 | b a n a n a s |
 1 | a n a n a s |
 2 | n a n a s |
 3 | a n a s |
 4 | n a s |
 5 | a s |
 6 | s |
The suffix array - a sneak preview

\[ T = b\ a\ n\ a\ n\ a\ s \]

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order.

0:  ```plaintext
   b\ a\ n\ a\ n\ a\ s
```
1:  ```plaintext
   a\ n\ a\ n\ a\ s
```
2:  ```plaintext
   n\ a\ n\ a\ s
```
3:  ```plaintext
   a\ n\ a\ s
```
4:  ```plaintext
   n\ a\ s
```
5:  ```plaintext
   a\ s
```
6:  ```plaintext
   s
```
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a} & \text{a} < \text{b} & \text{a}
\end{align*}
\]
Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{array}{c}
0 & b & a & n & a & n & a & s \\
1 & a & n & a & n & a & s \\
2 & n & a & n & a & s \\
3 & a & n & a & s \\
4 & n & a & s \\
5 & a & s \\
6 & s \\
\end{array}
\]

\[
\begin{array}{c}
a & a \\
< \\
b & a
\end{array}
\]
The suffix array - a sneak preview

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td>0</td>
</tr>
<tr>
<td>( a )</td>
<td>1</td>
</tr>
<tr>
<td>( n )</td>
<td>2</td>
</tr>
<tr>
<td>( n )</td>
<td>3</td>
</tr>
<tr>
<td>( a )</td>
<td>4</td>
</tr>
<tr>
<td>( a )</td>
<td>5</td>
</tr>
<tr>
<td>( s )</td>
<td>6</td>
</tr>
</tbody>
</table>

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order.

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
    a & < ba < bc
\end{align*}
\]
The suffix array - a sneak preview

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>2</td>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The symbols themselves must have an order

   throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{aa} & < \text{ba} < \text{bc}
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order
  throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a a} & \ < \ \text{b a} \ < \ \text{b c}
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes
lexicographically

- The symbols themselves must have an order
  throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
    a a & \ < \ b a \\
    b a & \ < \ b c \\
    b c & \ < \ b c a
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
& a\ a \ < \ b\ a \ < \ b\ c \ < \ b\ c\ a \\
& b\ a\ n\ a\ n\ a\ s \\
& a\ n\ a\ n\ a\ s \\
& n\ a\ n\ a\ s \\
& a\ n\ a\ s \\
& n\ a\ s \\
& a\ s \\
& s
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

- $a\,a < b\,a < b\,c < b\,c\,a$

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[ a\ a \ < \ b\ a \ < \ b\ c \ < \ b\ c\ a \]

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
    a \ a &< b \ a < b \ c < b \ c \ a \\
\end{align*}
\]

(in a ‘tie’, the shorter string is smaller)
Sort the suffixes lexicographically.

- The symbols themselves must have an order throughout we will use alphabetical order.

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
    a & a & < & b & a & < & b & c & < & b & c & a
\end{align*}
\]

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically.

- The symbols themselves must have an order throughout we will use alphabetical order.

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a a} & < \text{b a} < \text{b c} < \text{b c a} \\
\end{align*}
\]

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[ \text{a a} < \text{b a} < \text{b c} < \text{b c a} \]

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes

**lexicographically**

- The symbols themselves must have an order
  
  *throughout we will use alphabetical order*

just a fancy name for the order the strings would appear in a dictionary

In **lexicographical** ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a a} & < \text{b a} < \text{b c} < \text{b c a} \\
&(\text{in a ‘tie’, the shorter string is smaller})
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols themselves must have an order
  throughout we will use alphabetical order

just a fancy name for the order the strings would appear in a dictionary

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a a} & < \text{b a} < \text{b c} < \text{b c a} \\
\end{align*}
\]

(in a ‘tie’, the shorter string is smaller)

If the symbols don’t have a natural order, we use their binary representation in memory
The suffix array - a sneak preview

Sort the suffixes lexicographically

<table>
<thead>
<tr>
<th>T</th>
<th>b a n a n a s</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>b a n a n a s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a n a n a s</td>
</tr>
<tr>
<td>2</td>
<td>n a n a s</td>
</tr>
<tr>
<td>3</td>
<td>a n a s</td>
</tr>
<tr>
<td>4</td>
<td>n a s</td>
</tr>
<tr>
<td>5</td>
<td>a s</td>
</tr>
<tr>
<td>6</td>
<td>s</td>
</tr>
</tbody>
</table>
The suffix array - a sneak preview

Sort the suffixes lexicographically

\[
\begin{array}{cccccc}
T & b & a & n & a & n & a & s \\
0 & 1 & 2 & 3 & 4 & 5 & 6
\end{array}
\]

\[
\begin{array}{c}
1 & a & n & a & n & a & s \\
3 & a & n & a & s \\
5 & a & s \\
0 & b & a & n & a & n & a & s \\
2 & n & a & n & a & s \\
4 & n & a & s \\
6 & s \\
\end{array}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

Suffix Array

\[
\begin{array}{cccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

Suffix Array

\[ T = \text{bananaas} \]

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \]

\[ 1 \text{ as} \quad \text{a a a a} \]

\[ 3 \text{ a s} \quad \text{a a a} \]

\[ 5 \text{ s} \quad \text{a} \]

\[ 0 \text{ as} \quad \text{bananaa} \]

\[ 2 \text{ a s} \quad \text{a a a} \]

\[ 4 \text{ s} \quad \text{a a} \]

\[ 6 \text{ s} \]
The suffix array - a sneak preview

Sort the suffixes lexicographically
The suffix array - a sneak preview

Sort the suffixes lexicographically

The suffix array is much smaller than the suffix tree (in terms of constants)
The suffix array - a sneak preview

Sort the suffixes lexicographically

The suffix array is much smaller than the suffix tree (in terms of constants)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

To get the Suffix array perform a depth-first search (in lexicographical order)

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists.

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists.
- the $ is the smallest symbol in the alphabet.

To get the Suffix array perform a depth-first search (in lexicographical order).
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

To get the Suffix array perform a depth-first search (in lexicographical order)

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet
Constructing the Suffix Array from the Suffix Tree

$T$

```
bananas
```

Suffix Array

```
1 3 5 0 2 4 6
```

recall that we added a unique symbol $ to make sure the tree exists

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol \$ to make sure the tree exists.

- The \$ is the smallest symbol in the alphabet.

To get the Suffix array perform a depth-first search (in lexicographical order).
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

T  b  a  n  a  n  a  s
   0  1  2  3  4  5  6

Suffix Array
1  3  5  0  2  4  6

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

To get the Suffix array perform a depth-first search (in lexicographical order)

- recall that we added a unique symbol $ to make sure the tree exists
  - the $ is the smallest symbol in the alphabet
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

To get the Suffix array perform a depth-first search (in lexicographical order)

*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists.
- The $ is the smallest symbol in the alphabet.

To get the Suffix array perform a depth-first search (in lexicographical order).
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

this takes $O(n)$ time
• The (compacted) suffix tree of a (length $n$) text uses $O(n)$ space

• Finding all matches of a pattern $P$ of length $m$ takes $O(m + \text{occ})$
  
  where $\text{occ}$ is the number of matches

• Suffix trees can be built in $O(n)$ time

  but we have only seen the $O(n^2)$ time method

we assumed that the alphabet contains a constant number of symbols